DOCUMENT RESUME

ED 041 612 LI 002 078

AUTHOR	Glantz, Richard S.
TITLE	SHOEBOX: A Personal File Handling System for Textual Data. Information System Language Studies, Number 23.
INSTITUTION	Mitre Corp., Bedford, Mass.
REPORT NO	MTP-116
PUB DATE	Apr 70
NOTE	37p.
EDRS PRICE	EDRS Price MF-$0.25 HC-$1.95
DESCRIPTORS	*Computer Programs, Electronic Data Processing, *Filing, *Information Processing, Information Retrieval, *Information Storage, Information Systems, *Information Utilization, Input Output, Time Sharing

ABSTRACT

Until recently, the emphasis in information storage and retrieval systems has been towards batch-processing of large files. In contrast, SHOEBOX is designed for the unformatted, personal file collection of the computer-naive individual. Operating through display terminals in a time-sharing, interactive environment on the IBM 360, the user can browse through, edit, and reorganize passages of text. He can incorporate personal marginal notes and attach descriptors. Searching the files can be done on the basis of index terms or the pattern of the actual words in the text. Special emphasis is placed on mechanisms for treatment of synonyms and for system accommodation to user idiosyncrasies. (Author)

ED041612

INFORMATION SYSTEM LANGUAGE STUDIES NUMBER TWENTY THREE MTP-116

SHOEBOX: A Personal File Handling System for Textual Data

by

Richard S. Glantz

April 1970

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

THE MITRE CORPORATION
BOX 208
BEDFORD, MASSACHUSETTS

LI 002078

This document has been released for public dissemination.

MITRE Project 1249

ABSTRACT

Until recently, the emphasis in information storage and retrieval systems has been towards batch-processing of large files. In contrast, SHOEBOX is designed for the unformatted, personal file collection of the computer-naive individual. Operating through display terminals in a time-sharing, interactive environment on the IBM 360, the user can browse through, edit, and reorganize passages of text. He can incorporate personal marginal notes and attach descriptors. Searching the files can be done on the basis of index terms or the pattern of the actual words in the text. Special emphasis is placed on mechanisms for treatment of synonyms and for system accommodation to user idiosyncrasies.

ACKNOWLEDGEMENT

In the age of big science, progress is a group effort. Henry Bayard, Carter Browne, Stanley Cohen, Jeanne Fleming, Louis Gross, Ted Haines, Lewis Norton, and Donald Walker have all contributed to the design, implementation, and testing of the system. We also are indebted to Thyllis Williams of Inforonics, Inc., for guidance in both design and testing.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
DESIGN CONSIDERATIONS	3
BASIC OPERATION	7
HUMAN FACTORS	9
SPECIAL CAPABILITIES	13
A. Editing	13
B. Copying	14
C. Searching	16
D. Reorganizing	26
E. Input/Output	27
F. Housekeeping	27
HARDWARE AND SOFTWARE DETAILS	29
CURRENT DEVELOPMENT EFFORTS	31
REFERENCES	33

INTRODUCTION

The SHOEBOX system, a part of MITRE's long-term effort in the development of text-processing systems[1], is designed to be the electronic analog of a personal desk file drawer. A desk file drawer is conveniently at hand and readily accessible. It contains documents, reports, adversaria -- probably most of which, if the work of others, hasn't been thoroughly read, or if one's own work, remains unfinished. This material is organized under whatever whimsical scheme suits one's fancy; and as one's fancy changes, the file contents are variously combined or further segregated. (Or at least one would like to perform that kind of reorganization, at present a formidable undertaking.) Needless to add, the texts are set down in a variety of formats, the only common factor among them all being a close adherence to the grammar rules of natural language. Such an unstructured environment is the bane of digital mechanization, but it is to this problem that we have addressed ourselves.

The system described below is more than a passive repository for textual information. As an interactive, time-shared, computer-based system, it can serve a community of users, working privately or in concert. Each user can engage in such activities as browsing, searching, annotating, reorganizing, editing, indexing, and composing. He can digress at any time, perform another activity, and return to his original line of action. He can share his collection with others. He can apply one activity to the

results of another: a particularly good example, and one that is possible on few commercial information storage and retrieval systems, is performing a search on the results of a previous search.

Of course, the architecture of the computer and its peripheral gear impose some constraints on the form of the textual information, much the same as typewriter model and page size do in manual filing systems. SHOEBOX has three conventions: a maximum line length of 72 characters, no word division at the right margin, and typography coded as non-alphabetic prefixes to words (for those applications where it is relevant). No attention need be paid to page boundaries, which are, after all, an artifact of paper systems.

It must be emphasized here that SHOEBOX* is only a cover term. As relevant research products with the larger context of investigation mentioned at the beginning of this paper come to fruition, they are refined and incorporated into SHOEBOX. Thus the system is a continually evolving one, and this description of it is already obsolete. However, we do intend to maintain upwards compatibility.

Other systems oriented toward handling personal files are being developed by Engelbart[2], Nelson and van Dam[3], and Reitman[4].

---

* The name derives from the containers in which one of our projected users currently stores his vital information: they bear a striking resemblance to footwear packages.

DESIGN CONSIDERATIONS

SHOEBOX is designed to be especially accommodating to its human masters. Since the anticipated users of the system may have had no (professional) exposure to computing machinery, we deemed it important to ease the culture/technology shock. At the same time, we wanted to provide a versatile system capable of a wide range of text manipulations. Of equal importance, we wanted a system that could be transported to other computer centers. These overall considerations influenced our decisions in the realms of hardware configuration, software control, and data base structure.

The system is interactive, as mentioned earlier, to provide the user with immediate response to his action and to allow him to react immediately to that response. Displays were chosen for the remote terminals over typewriters or teletypes because of their ability to present large amounts of text almost instantaneously. In addition, their comparatively silent operation is more conducive to the human thought processes the system is supposed to enhance. For transportability, the system operates on the rather common IBM 360/50, under OS. The display consoles are IBM 2260's, whose prime attraction is their availability and relatively low price. We would prefer a larger capacity scope with some kind of cursor or position controller like a "mouse", but no acceptable versions were available at the inception of

this project, or are now, for that matter. Further details on the hardware and system software are discussed later in the paper.

From the user's point of view, the data base is a collection of files, each labelled with a mnemonic of 24 characters or less. (See Figure 1.) The files may be placed into a number of private file drawers, and each of these also has a mnemonic label. One file drawer, termed public, is available to anyone using the system.*

A file is a sequence of lines of text, which may be grouped into items. An item, which is typically several paragraphs in length, may have a one or two character category code associated with each line. The category code can be used to indicate titles, authors, dates, descriptors, personal annotations, or what-have-you. Although the user must identify which lines have which categories, the code scheme itself is completely his choice and need only remain consistent within any one file.

Meta-textual information can be coded as well. For example, if page reference to the hard copy original is needed, a line containing the page number and having the category "p" associated could be used at the change in page.

---

* There is also a system file drawer, which contains the programs SHOEBOX actually calls on to run itself. This arrangement makes modifying and improving the system quite convenient. Source programs are reorganized and manipulated in much the same way as straight text. This aspect of SHOEBOX will not be discussed here.

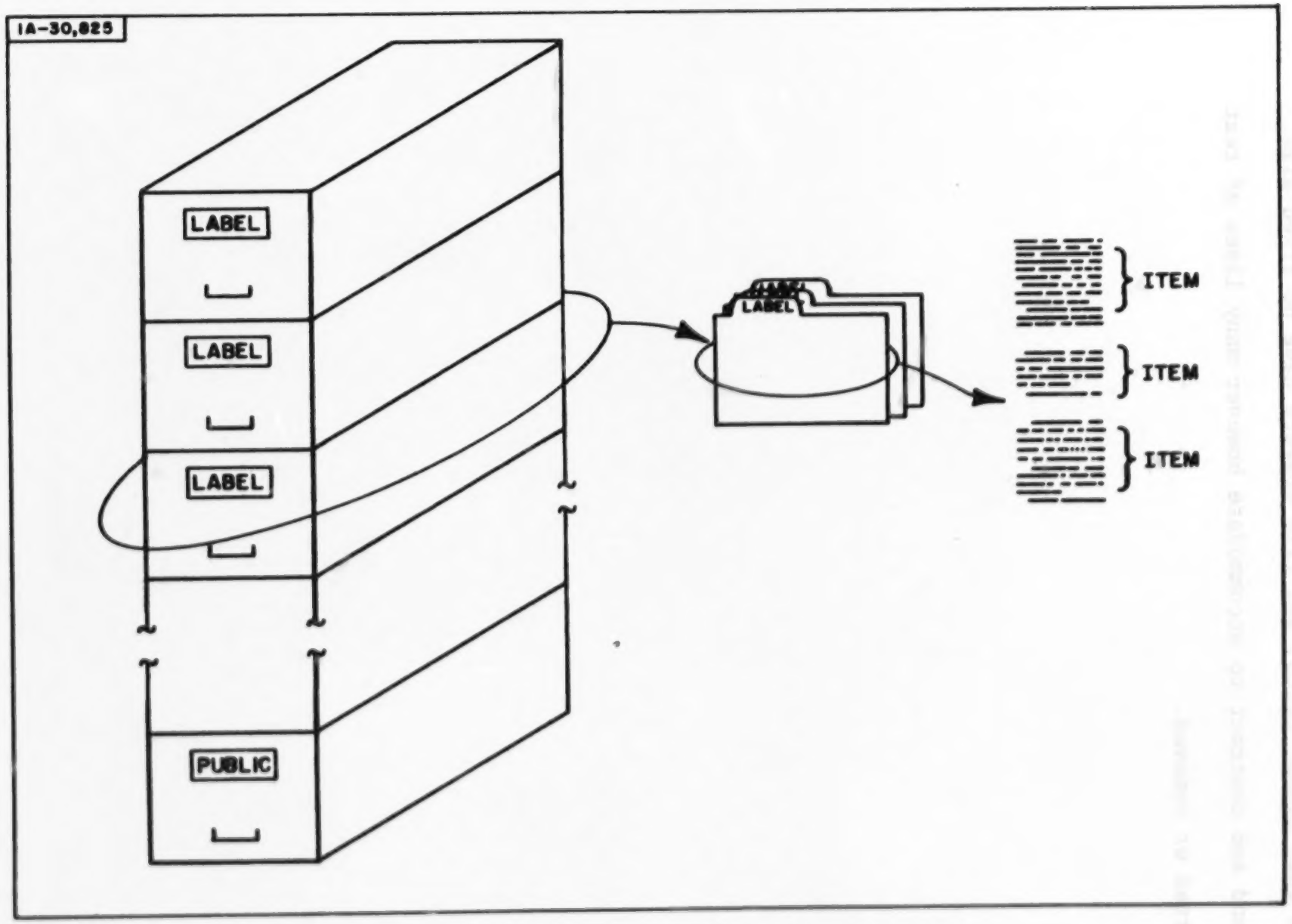
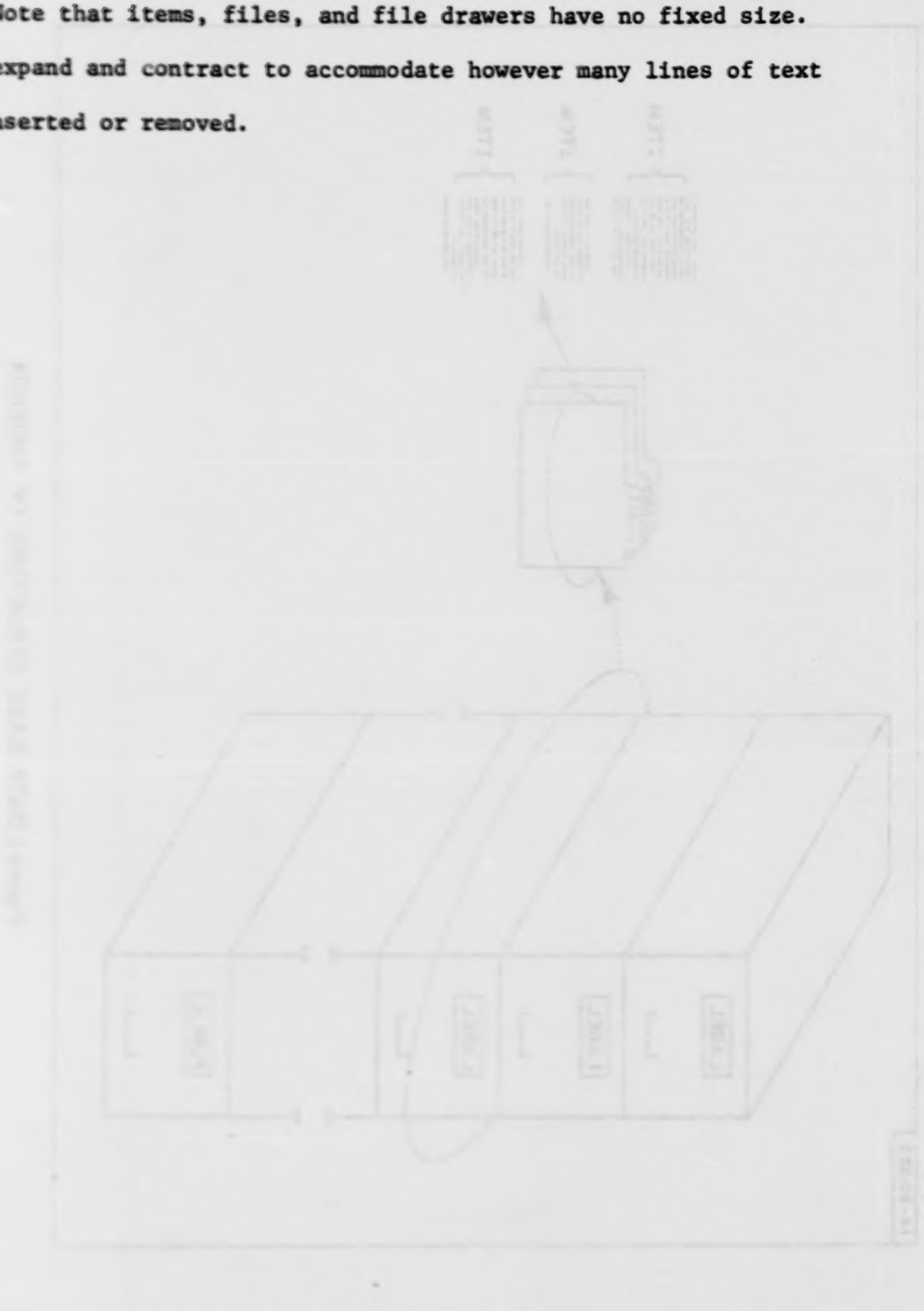


Figure I. DATA BASE STRUCTURE OF SHOEBOX

Note that items, files, and file drawers have no fixed size. They expand and contract to accommodate however many lines of text are inserted or removed.



BASIC OPERATION

To facilitate the detailed discussion which follows later in this paper, it will be helpful if we introduce at least the basic file manipulation operations.

After logging on to the system by identifying himself, the user opens the file drawer he intends to work with and selects a particular file to examine. Until he chooses another file, subsequent commands (with a few obvious exceptions discussed below) are considered to apply to the file he has "placed on his desk". The command EXAMINEFILE(irving), entered on the top line of the CRT, will select the file named "irving" and will display the initial lines of irving on the remainder of the scope.*

Beyond the right margin of each text line are displayed the line category, if any, and the line position in the file. Line positions are numbered sequentially upwards from 1; they are automatically maintained by SHOEBOX and need not be a concern of the user except for ontological reference.

To browse through irving, the user can repeatedly enter the command NEXT, which will step him through the file one scopeful at a time. The command PREVIOUS reverses the direction of travel. More rapid skipping around is achieved by entering an integer n, which will cause irving to

---

* We will adopt the convention henceforth of printing in full caps system commands and system responses.

be displayed beginning at the nth line. To jump directly to the last several lines of the file, the user enters LAST. Content-directed browsing is also possible. Suppose he is curious as to whether the description "fig newton" occurs later in the file. The command FIND ('fig newton') will cause SHOEBOX to search the rest of irving for the next line containing the character string "fig newton". The displayed response is either the relevant portion of irving or the comment NOTHING FOUND. This is a primitive kind of search; later in the paper we will discuss the more powerful search capabilities available in SHOEBOX.

Personal comments can be interspersed throughout a file. Whatever is entered below the response line on the display can be incorporated into irving with the command INSERTSCOPE. The new material will be treated the same as any other part of irving, although it would be judicious to mark the comment lines with a distinctive category code.

When the user is finished with irving, he can place it back in the file drawer with the command FILE. Alternatively, he can call out another file, say EXAMINEFILE(arnold), which puts irving away and places arnold on the desk.

With this brief introduction to the "flavor" of SHOEBOX, we would like to turn our attention first, to the very important human accommodation factors, and second, to a detailed exposition of the really powerful things the system can do.

HUMAN FACTORS

In the belief that an interactive computer system should be a complement to human thought rather than an intrusion, SHOEBOX commands are designed to mimic human textual activity. At the same time, actions which could be harmful if employed inadvertently are made somewhat difficult to perform.

One way we have made the system easy to learn is by grading the entire instruction set. The grading is done on the basis of both name and capability. For example, the novice may use the command EXAMINEFILE(irving) to call out the file labelled irving. After a brief exposure to the system, he would undoubtedly prefer using the abbreviated equivalent command, EF(irving). All long-named, frequently used commands have "obvious" abbreviations: IS for INSERTSCOPE, ML for MOVELINE, SF for SUBSTITUTEFILE, S for SUBSTITUTE, etc. More experienced users can step up to the level of extended commands, e.g., EF(irving 55) will bring irving out from the file drawer and display it beginning at line number 55. Thus the user can promote himself to advanced levels to match his own needs, work habits, etc. We are currently engaged in publishing a series of graded instruction manuals and reference guides to aid and encourage this kind of personal development.

Once learned, retention is high. The operations performed by the user, for the most part, parallel human desires, not machine requirements. The choice of command names and their verb or verb-noun construction is a familiar mnemonic device. Consistency also plays a role. The top line of the scope is reserved exclusively for user commands; the second line, for system response.

We have endeavored to adhere to the principle that for every user action, there is a machine response. Ordinary conversation between humans becomes trying unless the listener occasionally nods his head or murmurs some sort of acknowledgment. When the conversational partner is a computer, a lack of response or even a delayed response can be deadly. This is especially so for the intermediate-level user: the tyro has no expectations anyway and the sophisticate will sit and curse the system programmers, but the sophomore is the one most likely to panic and perform a disastrous action. After the user enters a command on the command line, SHOEBOX indicates that it has received the command by erasing the response line. If the command is incorrectly formulated, or if the request cannot be carried out for some reason, a comment identifying the difficulty is displayed on the response line. If the command is properly executed, either a confirmatory statement is displayed on the response line or a relevant portion of the file "on the desk" is displayed from the response line down. For commands which are likely to take over six seconds to carry

out completely, a running commentary of some reassuring sort appears on the response line.* †

The command set is designed to protect the user against himself. For example, wholesale deletion of material requires just a bit more thought. Unlike most operations on the "desk" file, the command to delete the entire file requires that the file be named specifically. In another situation, the unsophisticated user is not told how to directly replace the contents of one file with the contents of another. While such a command does exist, its hasty application could cause the unwanted destruction of the target file. Instead, the inexperienced user is taught to first <u>append</u> the source file to the target file and then delete that portion of the target file he doesn't wish to save. By requiring such deletion actions to be <u>explicit</u> on the part of the user, we hope to save him from unnecessary grief.

Perhaps the ultimate in user accommodation is reached with the facility provided in SHOEBOX for coining personal alternates for any of the commands. One of the staff members of the project, for instance, feels happier with EDFIL instead of EXAMINEFILE, STORP instead of SUBSTITUTE, and ADSCOPE instead of INSERTSCOPE. Any such neologism

---

* At the time of this writing, we have not fully implemented this feature: it appears only on search commands at present, as a rendering of the number of (1) the current line being searched (in multiples of 20, to avoid overdemand on the central processor) or (2) the latest line where a match was found.

† See Miller[5] for an extended discussion of responsiveness in interactive systems.

can be defined in the system lexicon with a single command. Furthermore, because the lexicon will handle any number of alternates to a SHOEBOX command name, potentially every user could satisfy his iconoclastic desires.

We all make mistakes. If the user wishes to prevent SHOEBOX from completing the execution of his last command, he hits the "enter" key on the console keyboard again. Control then pops up to the neutral, or waiting state. At this point, the user can return the file, unaffected, to the file drawer with the command GIVEUP. Or, he can change relevant variables (e.g., output format for a PRINTF command) and resume execution with the command ENDWAIT.

When SHOEBOX itself is suspect, the command ? will cause the system to interrogate itself, indicate what its state of activity was, and return to that state. Sample responses to ? are WORKING, WAITING (for a command), and SWAPPING (between users). In case of system failure, the response to ? is a locked keyboard.

SHOEBOX is accommodating up to a point. At twenty errors, it becomes intolerant and logs off.

SPECIAL CAPABILITIES

A. Editing

Correcting typographical errors, rewording phrases, and inserting and deleting passages are typical editing operations on a text. With SHOEBOX, the author can immediately inspect the results of his decisions. If he remains dissatisfied, he can try again, or he can put the questionable material aside and come back later, or he can make a copy of it for subsequent comparison with alternate phrasings.

There are two basic ways to effect local changes in the stored text. One is to make use of the display hardware and type over the displayed lines of text. When the command SUBSTITUTE is given, the text lines on the scope replace the corresponding lines in the file. As a check, SHOEBOX redisplays the same scopeful, but from the now modified file.

The second way makes use of a search-and-replace command called CHANGE. The command CHANGE('a' 'b' n) replaces the string of characters a in line n of the desk file with the string of characters b. If there is more than one occurrence of a, only the first will be affected. Taking the previous sentence as an example and with the correct choice of n, CHANGE('will be' 'is' n) will transform the consequent phrase to read, "only the first is affected."

With its ability to insert arbitrarily long strings, CHANGE can affect as well the text outside line n. On the specified line, deletions are closed up; and insertions are spilled over to a new line (and line numbers greater than n are incremented accordingly to allow for the

interlineation). The contractions and expansions caused by a CHANGE command are not bubbled through the entire file. That would degrade the response time. Instead, when he has completed his editing modifications, the user can request that the file be PACKed. This operation is analogous to the request one makes of a secretary to "type over" a manuscript; however, with an electronic file, the redoing takes only seconds.

CHANGE is a graded command. If a second numerical argument is specified, as in CHANGE('a' 'b' m n), the first occurrence of a, if any, in line m, line m+1, ... line n is replaced by b. The command CHANGEALL makes the same replacement for every occurrence of a. Multiple changes within a line can be achieved by providing successive pairs of strings in the argument, as in CHANGE('a' 'b' 'c' 'd' m n). Another CHANGE-type command affects the line category code only.

An extensive error analysis is provided which informs the user of an error in syntax, the non-occurrence of a in a line, the non-occurrence of a line number in the file, and for the wise guys, the requirement that m cannot be greater than n nor can either of them be less than one or greater than the number of lines in the file.

B. Copying

Copies of items or of entire files can be made and placed at the end of any file or within any file drawer. For safety's sake, however, no command can make reference to a particular place within a file unless that file is on the desk (so that the affected portion can be

visually inspected). Thus, only portions of the desk file can be copied and placed at specified points within the desk file.

The commands which copy from the desk file are exemplified by the following:

APPENDTO(arnold 34)--places a copy of line 34 of the desk file at the end of arnold (which can also be the desk file)

INSERTLINE(12 34) -- places a copy of line 34 of the desk file before line 12 of the desk file

In the first example above, if arnold doesn't exist in the file drawer, a new, empty file with that name will be made; and then the appending operations will be performed. Items as well as individual lines can be copied: APPENDTO(arnold 34 52) will copy the item encompassed by lines 34 through 52 to the end of arnold. INSERTLINE(12 34 52) would be processed in a similar fashion. Further, extensions of these commands allow several non-contiguous passages to be copied at the same time. In this manner, an extract of an article could be taken, and stored in a file of such extracts or tacked onto the article itself. The command INSERTFILE, with appropriate arguments, will copy an entire file and place it before a specified line of the desk file.

To make a separate copy of a whole file, there is no need to take the file out of the file drawer. The command COPYF(irving arnold) makes a copy of irving and labels it arnold. If there already is an arnold in the same file drawer, its contents are discarded first. Because of the danger of catastrophe with this command, the incautious user is well advised to employ instead the following two-step procedure:

```
EXAMINEFILE(irving)
APPENDTO(arnold 1 LAST)
```

If there was something in arnold beforehand, the appending operation will preserve it. An alternate safe procedure is to enter first

EXAMINEFILE(arnold)

and then, only if the response is ARNOLD NOT A FILE, enter

COPYF(irving arnold)

The command APPENDFILE(irving arnold) makes a copy of irving, if it exists, and appends it to the end of arnold, if it exists. Thus, a third alternative is

APPENDFILE(irving arnold)

C. Searching

While rapid access and facile manipulation materially aid the collector and analyst of information, the crux of the problem is discovery and rediscovery of what is pertinent. There is always an uncomfortable feeling about pigeonholing material under headings or even tagging material with a multitude of descriptors--critical snips of information seem to get lost by this precategorization.

The situation is especially acute in public affairs, where personalities, events, and issues which received little notice in the past may suddenly move to center stage. When past history lies scattered among the files, retrieval is difficult. The ability of SHOEBOX to search on the actual words in the texts, as well as on descriptors and categories, greatly enhances the reliability of the search.

We have already introduced our most primitive search command, FIND. Briefly, FIND ('fig newton') will search the desk file, from the second text line currently displayed to the end of the file, for the first occurrence of the string "fig newton" which lies completely within a single text line. (At the request of some users, we have implemented a related command, FINDANCHORED, which looks only along the left margin.)

A higher degree of search capability is attained with SIMPLESEARCH. When this command is entered, the following questionnaire appears on the scope in response:*

```
AFTER COMPLETING THIS QUESTIONNAIRE, HIT "ENTER".
..................... IS THE FILE YOU WANT SEARCHED.  TYPE SEARCH REQUEST BELOW:

....................... ARE THE CODES OF THE LINES YOU WANT SEARCHED IN THE FILE.
..................... IS YOUR NAME FOR THE FILE WHERE THE ANSWERS WILL BE PUT.
..... ARE THE NUMBER OF LINES BEFORE THE MATCH LINE THAT YOU WANT IN EACH ANSWER
..... ARE THE NUMBER OF LINES AFTER THE MATCH LINE THAT YOU WANT IN EACH ANSWER
... DO YOU ALSO WANT THE DATE OF THE RELEVANT DOCUMENT ALONG WITH EACH ANSWER?
```

Note that: (1) the file to be searched need not be on the desk

(2) the entire file is searched

(3) the search can be restricted to lines with certain categories

(4) the search does not cease at the first match, but begins again at the next item

(5) the "answers" are not displayed but put into a file for subsequent inspection and manipulation

(6) pre- and post-context can be specified (with FIND, the pre- and post-context are fixed at 0 and 9 lines, respectively)

---

* Not explicitly called out in the questionnaire are the values for two system variables, which indicate (1) the category assigned to the date line (or any other category line to introduce each answer) and (2) the category of the line which begins or terminates every item. We assume these variables, once set by the user, are uniform for his entire file collection.

We will describe the form of the search request by illustration rather than by formal definition. In this instance, a formal statement would be pedantic. The reader is first advised to consult Table 1.

As an example, let us suppose an analyst keeps a file of newswire reports on South Vietnam. Let us suppose further that many months later he is assigned a task which involves retrieving those news bulletins which give actual figures in mentioning American troop withdrawals. One of the search prescriptions he could try is:

'*000' Exs 'troop' 3 'withdr' | ('pull' out)

Reports containing any of the following phrases would be identified and retrieved:

...30,000 US troops withdrew...
...5000 US troops will withdraw...
...46,000 troops will be pulled out...
...10,000-man troop withdrawal...
...365,000 troops could have been withdrawn...
...44,000 troops to pull out...

It should be noted here that, under SIMPLESEARCH, unlike FIND, these phrases can be found even if they begin and end on different lines.

After the search has been completed, the results and a copy of the questionnaire itself are placed in the designated answer file for examination. Consequently, the search can be reformulated and applied to the source file or the answer file. If a third file is designated as the repository for the second search, the two results can be compared. Or, the same search can be applied to another file without rekeyboarding anything except the file label. Finally, using the annotation and

Concept	Example	Searches for all occurrences of
Word	newton	newton, newton's, newton?, newton.", 3newton (supposing the prefixed numeral denoted typographical coding), 3newton-3raphson, newton-sec2/m; i.e., any matching string of characters delimited by non-alphabetics
Concatenation	fig newton	fig newton, fig newton's, "fig" newton, etc.
Disjunction	fig | date	fig, date, fig's, date? etc.
Specific Optionality	newton %interference rings	newton interference rings, newton rings, etc.
Complementation	newton -interference rings	newton diamond rings, newton pastry rings, newton circus rings, etc., but not newton rings or newton interference rings
Phrasing	date | (fig newton) date | fig newton	date, fig newton, etc., but not date newton; date newton, fig newton, etc., i.e., disjunction takes precedence over concatenation (and over specific optionality and complementation, as well)
Anchored Stem	'newton*'	same as the word newton, plus newtonian, newtonians, newtons, and (newtonian), newtons', etc.
	'*newton'	same as the word newton, plus antinewton, etc.
Free Stem	'*newton*'	both the above, plus antinewtonian, etc.
	'newton'	equivalent notation
	'3newton'	3newton, 3newton's, etc., but not newton, newton's, etc.
Non-specific Optionality	newton 1 rings	newton rings, newton interference rings, newton diamond rings, newton pastry rings, etc.
	newton 7 rings	newton, etc., followed by up to 7 intervening words and stems, followed by rings, etc.

Table 1. Composition Possibilities for SIMPLESEARCH

reorganization commands, our Vietnam analyst can compose a report and incorporate the corroboratory search results.

A third degree of search capability is reached with CSEARCH (for contingent or complex search). This search request provides greater selectivity over the choice of items to be searched within the file and allows greater control over the format of the answers. Especially potent is the search prescription itself: it allows unordered search terms, contingent searches, and synonymic substitution.

The command CSEARCH causes the following questionnaire to be displayed:

```
**NAME OF FILE->                                                          1
**SEARCH FROM LINE->      **TO LINE->                                     2
**REMEMBER LINE CODES->(           ) **IGNORE->(     ) **TERMINATOR->     3
**SEARCHREQUEST ('LINE CODES' 'SPECIFICATION'; ...;)-> (                   4
                                                                          5
                                                                          6
                                                                          7
                                                                          8
                                                                          9
                                                                )         10
```

(The line numbers are displayed to allow modifying the answers to the questionnaire with the SHOEBOX editing commands.) Line 1 asks for the name of the file to be searched; line 2 allows the user to limit the extent of the search; line 3 provides for the setting of certain system variables; and lines 5 through 10 accept the search prescription. Table 2 lists the additional composition possibilities under CSEARCH.

Conjunction is typically used when dealing with key words or descriptors. For example, the search prescription

(k d) wood, preservative

Concept	Example	Searches for all occurrences of
Conjunction	date, fig	both date, date), etc. and fig, fig/, etc. anywhere in same text portion
	date, fig newton	both date, etc. and fig newton, etc., i.e., concatenation takes precedence over conjunction
Numerical Interval	$NUM('69.01' '70.03')	items dated between January, 1969 and March, 1970 [see text for further explanation]
Substitution	$CLASS snacks	the components dictated by the search pattern labelled "snacks" [see text for further explanation]

Table 2. Additional Compositional Possibilities for CSEARCH

would be expected to find items relating to preservatives for wood, where d and k are the categories of the lines containing the index terms.

$NUM is a two-argument function which can be embedded in a search specification. To illustrate with a simple usage, let us assume that the entire search prescription is of the form:

c $NUM('m' 'n')

where c is a category code and m and n are non-negative integers (with or without internal punctuation; e.g., 52,500, 69.03.14, 3.14159). Then any item whose c line has a number in the closed interval [m,n] will be found and selected (provided c, m, and n have a consistent format). This numerical capability has many applications; the example shown in Table 2 for calendar dates is particularly appealing.

One of the most attractive facilities within CSEARCH is the power of pattern substitution. Consider our analyst again. Suppose

now he is given the assignment of accumulating all reports which mention cease-fire violations in the Middle East. He might begin by formulating a tentative search specification

```
arabs 3 'attack' israel
```

which will find sentences such as "Arabs attacked Israel last night". But since many nations fall under the rubric "arab", he will soon realize that a better specification is

```
jordan | egypt | lebanon | syria | iraq | (saudi arabia)
3 'attack' israel
```

This phrasing will find "Jordan has attacked Israel", etc. Using stems instead of words, as in

```
'jordan' | 'egypt' | 'leban' | 'syria' | 'iraq' | (saudi 'arabia')
3 'attack' 'israel'
```

he can locate "Jordanian troops attacked Israeli positions", "Lebanese artillery attacked Israeli fortifications", etc. Rather than have to rekey this representation for "arabs" each time he modifies his search request, the analyst can define them to be members of a labelled set. Then he would write the previous search specification as

```
$CLASS arabs 3 'attack' 'israel'
```

where he has defined $CLASS arabs, after further thought, to be the long disjunction

```
'arab' | 'guerilla' | (al fatah) | 'jordan' | 'egypt' | 'leban' |
'syria' | 'iraq' | (saudi 'arabia') ...
```

In this manner, he has defined the set of entities he can henceforth refer to as "arabs". This maneuver not only saves him from keyboarding this long pattern again as he constructs related search prescriptions,

it also saves him the bother of remembering or redetermining what political groups of people comprise Arabs in the Middle East. Thus he can readily specify the reverse request,

```
'israel' 3 'attack' $CLASS arabs
```

"Attack", of course, is not the only verb which connotes aggressive activity. The analyst could again reformulate his search specification as

```
$CLASS arabs 3 $CLASS aggress 'israel'
```

where he has defined $CLASS aggress to be the pattern

```
'attack' | 'bomb' | 'shell' | 'strik' | struck | 'hit' |
'straf' | 'raid' | 'sink' | sank | sunk
```

These class definitions become a permanent part of the user's file collection. At any time, however, he can employ the editing commands of SHOEBOX to examine the CLASSes he has defined and to effect a change in content or label.

$CLASSes can be embedded in one another. For example, journalistic style allows such synecdochic expressions as "Amman claims their artillery shot down..." Our analyst might very well wish to set up equivalences between country and capital, viz.:

```
$CLASS israel  := {'israel' | (tel aviv)}
$CLASS jordan  := {'jordan' | amman}
$CLASS egypt   := {'egypt' | cairo}
$CLASS lebanon := {'leban' | beirut}
$CLASS syria   := {'syria' | damascus}
$CLASS iraq    := {'iraq' | baghdad}
$CLASS saudi   := {(saudi 'arabia') | ----*}
```

---

* The capital of Saudi Arabia is left as an educational enterprise for the reader.

and in turn:

```
$CLASS arab := {arab | guerrilla| (al fatah) | $CLASS jordan |
                $CLASS egypt | $CLASS lebanon | $CLASS syria |
                $CLASS iraq | $CLASS saudi}
```

and perhaps even further:

```
$CLASS mideast := {$CLASS arab | $CLASS israel}
```

Concomitant with the power of embedded $CLASSes is the danger of self-referral. CSEARCH expands all $CLASSes in the search prescription given it before actually proceeding with the search. If there is an infinite loop, or if the expanded search prescription is so huge that it threatens to take up all the allotted list space in the computer, the comment VISCIOUS LOOP is displayed on the response line of the scope.

The concept of $CLASS is also useful in dealing with spelling variations in transliteration. The name of the leader of Al Fatah has appeared in print as "Yasser", "Yassir", and "Yasir". Often one entity has several names: trinitrotoluene, TNT; lysergic acid diethylamide, LSD, LSD-25; Songmy, Song My, Son My, My Lai, Mylai, Mylai #4, Mylai Village #4, Pinkville. Establishing a substitution class both records the alternations and ends the reliance on one's memory.

The indolent and the impatient can also use $CLASS to advantage. Why keyboard "Society for the Prevention of Cruelty to Animals" if a substitution class with "spca" can be set up? If there is no standard acronym, make up your own nickname:

```
define $CLASS rocky := {governor nelson h. rockefeller | nelson
                        rockefeller | the new york governor}.
```

allows him to label the answer file, to store with the answers the original search questionnaire with the $CLASS terms in the search prescription expanded out, and to control the formatting of the answer file and certain associated data.

D. Reorganizing

Armed with the results of his searching, the user can make new files, discard unwanted files, merge files together, rearrange items within a file, and move files from one file drawer to another. For the author, reorganizing commands can be used effectively to cut-and-paste portions of text in writing reports. Indeed, in organizational structures where reports are passed from the lowest stratum to the highest level through a series of middle management filters, SHOEBOX offers a convenient way to extract data from an incoming report, add salient observations of your own, and pass the "new" report on to the next higher level.

The items within the desk file can be placed in ascending or descending order with the command ORDER. Among the arguments to this command is a sequence of {category code, sort key} pairs. If, for any item, the first pair cannot apply because the code and key do not co-occur in that item, the second pair is tried, and so on. This feature is useful, for example, if we wished to arrange a file of documents by author, or, if none, by editor. By employing ORDER several times, one can organize the items in a file, say, first with respect to date; and within date, by author; and within author, by title.

When first conceived, $CLASS was seen as a means to facilitate synonym sets and thesaurus hierarchies (and hence the name, $CLASS). As programmed, however, it is an instrument for wholesale pattern substitution. Any pattern, even a search prescription itself, can be defined as a $CLASS. Thus, a frequently required search prescription can be incorporated within a $CLASS and called out for subsequent use.

There is one search mode in CSEARCH left to discuss--contingency. If a sequence of search prescriptions is given, separated by semi-colons, the subsequent search prescription only applies to the items satisfied by the previous prescription. While effectively only a conjunction of searches, this mode of search will save a considerable amount of search time in large files. By way of illustration, the following search prescription

```
    d  $NUM('69.06.20' '69.08');
    a  south vietnam;
  (h b) '*000' %$CLASS us 'troop' 3 'withdr' | ('pull' out);
```

will first, select from the file all items whose d line contains a date between June 20, 1969 and the end of August, 1969; second, select from those items all items whose a line contains the descriptor "south vietnam"; and third, select from those items all items whose h or b lines contain a text passage satisfying the search specification shown.

At the completion of the search, a line number reference to each "hit" and the total number of "hits" are displayed. Based on this information, the user may decide to broaden, narrow, or reformulate his query. Or he may decide to place the items found by CSEARCH in an answer file. To do this, he completes another questionnaire, which

Items can be discarded from the desk file with the command REMOVELINE(m n), where m and n are the line numbers delimiting the unwanted passage. Alternatively, they can be shifted to the end of another file, say irving, with REMOVETO(irving m n). Or, they can be shifted to another position within the desk file, using MOVELINE(k m n), where k denotes the line number before which the passages are to be inserted. Of course, if m=1 and n=LAST, the entire file is affected.

Comments keyboarded on the display may be inserted into the desk file with INSERTSCOPE (k); or they may actually replace specified lines in the desk file, using SUBSTITUTESCOPE(m n). Scope comments can also be appended to the end of any file, say arnold, with the command APPENDSCOPETO(arnold).

E. Input/Output

A hard copy of irving can be printed offline with the command PRINTF(irving); similarly, PUNCHF(irving) punches irving onto cards. Other commands allow the reading and writing of files from and to seven- and nine- track tape or any OS sequential data set. For example, we have successfully passed files from SHOEBOX to IBM's Conversational Programming System (CPS) and vice versa; we also have passed files to a KWIC program.

F. Housekeeping

All files are automatically date-stamped by SHOEBOX with the day/month/year that the file was last changed in any way. When a file is brought out for display, this information, together with the name and size of the file, is displayed on the response line.

If the user wishes to see all the file labels in his collection, he enters the command LISTF( ). The argument within the parentheses determines the file drawer; if there is no argument, all file labels for both public and private file drawers are displayed.

For both training and system debugging purposes, a record is made of all commands and other material entered into the system at the display keyboard and all displayed responses by the system. The hour/minute/second when each command was entered is also saved. This record-keeping function can be partially or totally deactivated at log-on time. Otherwise, the record accompanies the off-line printout for each user.

HARDWARE AND SOFTWARE DETAILS

SHOEBOX is implemented on the IBM 360 under release 18 of OS/360 (PCP, MFT, or MVT), in a 225K high- or low-speed partition, and with a private IBM 2316 disk pack. If SIMPLESEARCH and CSEARCH are not needed, it will operate in 140K. The system as presently implemented will support up to eight IBM 2660 Model 1 (12x80) display stations in a local configuration supported by the Graphics Access Method (GAM). Budget considerations, however, have limited our work experience to four stations. These terminals can be located up to 2000 feet from the display controller. The transmission speed is quite fast: 2500 characters per second, or on the order of half a second to write the entire scope.

SHOEBOX is written in TREET, a list processing language[6] derived from LISP; and the system is run under the TREET/360 operating system. The TREET filing system operates, using OS direct access techniques, by dividing a single OS sequential file into small individual segments called pages, which are dynamically allocated and released. A SHOEBOX file is an ordered set of these pages. (The line numbers of the file are not stored with the file, but they are generated as required.) In this manner, modification of the file does not entail rewriting the entire file, but only the pages affected and the file directory. The directory is hash-coded, so that file access time is nearly independent of the number of files.

The approximate capacity of the system is 100 to 300 file labels per file drawer (depending on label length) with direct access to 2000 double-spaced typewritten pages of text per file drawer. The number of file drawers is dependent on the amount of space available on direct access devices.

Initial response time to a request is on the order of a few seconds. Time to process completely the request is a function of whether other jobs are utilizing the central processor or the input/output equipment, whether we are given a partition in high- or low-speed core, and the complexity of the request.

CURRENT DEVELOPMENT EFFORTS

In the SHOEBOX system described here, searching within a line is carried out sequentially, character by character. An experimental system has been programmed to test the potential savings of an index search. That is, for each file, an alphabetized list of its words was compiled, together with a set of pointers to their locations in the text. There are two advantages to searching the index instead of the text itself: (1) the time of search is more-or-less independent of the size of file, and (2) the search, being sentence-oriented, takes cognizance of natural text boundaries; e.g., a request for the phrase "fig newton" won't deliver the item containing the passage, "The Birge-Vieta method isn't worth a fig. Newton-Raphson isn't worth much either." There are compensating difficulties: (1) what algorithm determines sentence boundaries (recall that a period has more than the terminal function)?, (2) how should right-anchored stems occurring in the search prescription be handled?, and (3) must the index be recompiled after every modification to the file? Nonetheless, our findings with this experimental program showed savings in search time of an order of magnitude. With this encouragement, we are in the midst of incorporating a similar index search in SHOEBOX.

For use in a community of research workers, we would like to implement a "mailbox" feature. At log-on, the individual's name would cause

any messages left for him by others to appear on the display. A less intrusive alternative would be for a message like SOMETHING IN YOUR MAILBOX to appear.

Other projects in the early design stages are (1) graded error messages, (2) structured file drawers, and (3) statistical capabilities.

Under consideration is a facility for designating one member of a synonym set as having "print" status. This option would be especially welcome in composing reports: the author could keyboard his personal acronym or nickname for a wordy phrase or hard-to-spell proper noun, but the display and the file would contain the "official" forms.

It is our intention to incorporate into SHOEBOX increasingly sophisticated linguistic techniques of the kind represented in SAFARI[7], a MITRE text processing system in which the information content of English sentences is stored and retrieved directly.

REFERENCES

1. D. E. Walker "Computational Linguistic Techniques in an On-Line System for Textual Analysis" MTP-105, The MITRE Corporation, July 1969.

2. D. C. Engelbart and W. K. English "A Research Center for Augmenting Human Intellect" Proc. FJCC (1968) 395-410.

3. S. Carmody, W. Gross, T. H. Nelson, D. Rice, and A. van Dam "A Hypertext Editing System for the /360" in M. Faiman and J. Nievergelt (ed.) Pertinent Concepts in Computer Graphics Univ. of Illinois Press, 1969.

4. W. Reitman, R. B. Roberts, R. W. Sallvain, D. D. Wheeler, and W. Lin "AUTONOTE: A Personal Information Storage and Retrieval System" Proc. 24th Natl. Conf. ACM (1969) 67-75.

5. R. B. Miller "Response Time in Man-Computer Conversational Transactions" Proc. FJCC (1968) 267-277.

6. E. C. Haines "TREET, A List Processing Language and System" MTP-104, The MITRE Corporation, March 1969.

7. L. M. Norton "The SAFARI Text Processing System: IBM 360 Programs" MTP-103, The MITRE Corporation, September 1968.